Algoritmos y Estructura de Datos: Examen 3 (Solución)

Grados Ing. Inf. y Mat. Inf. Enero 2011

Departamento de Lenguajes, Sistemas Informáticos e Ingeniería de Software

Apellidos: .....

Nombre: .....

DNI / NIE: ..... **Núm. matrícula:** .....

Normas

- Este examen consta de **6 preguntas** en **3 páginas**.
- La puntuación total del examen es de **10 puntos**.
- La duración total del examen es de **90 minutos**.
- El examen debe contestarse **en las hojas que se proporcionan**.
- Deben rellenarse los campos obligatorios **apellidos, nombre, y DNI/NIE**.
- Las calificaciones provisionales de este examen se publicarán en el Aula Virtual el **26 de enero de 2012** junto con las soluciones. La fecha de la revisión de este examen es el **30 de enero de 2012**. La hora y lugar de dicha revisión se anunciará en el Aula Virtual.
- En las preguntas con varias opciones, **sólo hay una respuesta válida por pregunta**. En este caso toda pregunta en que se marque más de una respuesta se considerará incorrectamente contestada y toda pregunta incorrectamente contestada restará del examen una cantidad de puntos igual a la puntuación de la pregunta dividido por el número de alternativas ofrecidas en la misma menos uno. Es decir, una respuesta incorrecta en una pregunta de un punto con cuatro alternativas resta $\frac{1}{3}$ de punto.

(1 punto) 1. **Se pide:** Indicar de entre las siguientes posibilidades la definición correcta de doble dispersión («double hashing»):

   (a) Método de sondeo para tablas de dispersión con resolución de colisiones mediante encadenamiento («separate chaining») que utiliza una función de dispersión auxiliar en el cálculo de la siguiente posición de sondeo.

   (b) Método de sondeo para tablas de dispersión con resolución de colisiones mediante direccionamiento abierto («open addressing») que utiliza una función de dispersión auxiliar en el cálculo de la siguiente posición de sondeo. ✔

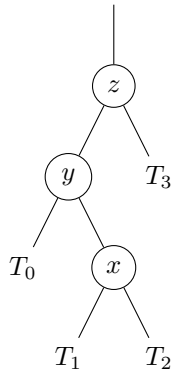
   (c) Método de sondeo para tablas de dispersión con resolución de colisiones mediante encadenamiento («separate chaining») que utiliza la función de dispersión dos veces en el cálculo de la siguiente posición de sondeo.

   (d) Método de sondeo para tablas de dispersión con resolución de colisiones mediante direccionamiento abierto («open addressing») que utiliza la función de dispersión dos veces en el cálculo de la siguiente posición de sondeo.

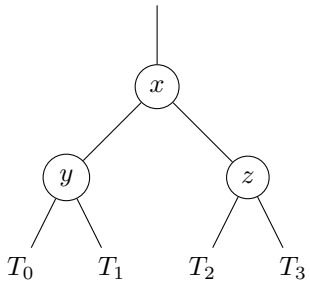
(1 punto) 2. Se tiene una tabla de dispersión con direccionamiento abierto («open addressing») implementada internamente usando un vector («array») de Java de tamaño N fijo. **Se pide:** Dar una cota superior para el factor de carga, razonando la respuesta.

> El factor de carga n/N es menor o igual que 1 pues sólo puede haber una entrada en cada posición del vector y cuando la tabla está llena $n = N$ y por tanto $N/N = 1$.

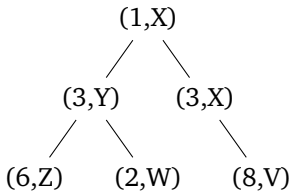
(1 punto) 3. Se tiene el siguiente subárbol de un árbol AVL en el que z es un nodo desequilibrado:



Se pide: Dibujar el subárbol resultante después de realizar el equilibrado.



(2 puntos) 4. En siguiente árbol almacena entradas en sus nodos con claves que son enteros. **Se pide:** Indicar si el siguiente árbol representa un montículo, razonando la respuesta tanto en caso afirmativo como negativo.



El árbol no es un montículo por dos razones:

1. No es un árbol (casi)completo pues el único hijo del nodo (3,X) es un hijo derecho.
2. En los nodos (3,Y) y (2,W) no se cumple la propiedad «heap-order».

(2 puntos) 5. Se dispone de una implementación de colas con prioridad mediante montículo que utiliza los siguientes atributos:

```
protected Entry<K,V> a[];       // Vector
protected int numElements;      // Número de entradas en el vector
protected Comparator<K> comp;   // Comparador de claves.
```

Los métodos `parent`, `left`, `right`, y `swap` ya están implementados. Los tres primeros toman la posición de una entrada y devuelven la posición de su padre, su hijo izquierdo, y su hijo derecho respectivamente. El método `swap` toma dos posiciones como parámetro e intercambia las entradas almacenadas en dichas posiciones del vector.

Se pide: Implementar en Java el método `protected void upHeap(int pos)` que implementa el «up-heap bubbling» del montículo, es decir, que comprueba y reestablece la propiedad de «heap-order» cuando se ha insertado un nuevo nodo en el montículo. El método `upHeap` es invocado por `put` pasando como parámetro la posición en el vector `a` del nuevo nodo insertado.

```
protected void upHeap(int pos) {
  while( pos>1 &&
         comp.compare(a[pos].getKey(),a[parent(pos)].getKey()) < 0 ) {
    swap(currentPos, parent(currentPos)) ;
    pos=parent(pos) ;
  }
}
```

(3 puntos) 6. Se dispone de una implementación de tablas de dispersión con direccionamiento abierto («open addressing») que usa los atributos:

```
private int numElements;      // Número de entradas en la tabla
private Entry<K,V> table[];   // Vector tabla
```

así como el método `int hashValue(K k)` ya implementado que devuelve un entero entre 0 y `table.length-1`. **Se pide:** Implementar el método `private int findEntry(K k)` que busca la entrada con clave `k` en la tabla usando sondeo lineal. Si dicha entrada está en la tabla entonces el método debe devolver la posición de la misma en el vector. Si no está en la tabla entonces el método debe devolver un valor negativo.

```
private int findEntry(K k) {
  for ( int startPos  = hashValue(k),
        int pos       = startPos ,
        boolean found = false ,
        boolean end   = false ;
        !found && !end ;
        pos = (pos+1)  % table.length )
  {
    end   = table[pos] == null || pos == startPos ;
    found = table[pos] != null && table[pos].getKey() != null
                               && table[pos].getKey().equals(k) ;
  }
  // Either 'found' is true, or 'end' is true, or both.
  return (found && !end) ? pos : -1 ;
}
```